(1)

Working Paper No. 362

Contract N00014-85-K-0143

REUSING STRUCTURED MODELS VIA MODEL INTEGRATION

by

Arthur M. Geoffrion

September 1988

WESTERN MANAGEMENT SCIENCE INSTITUTE
University of California, Los Angeles

WESTERN MANAGEMENT SCIENCE INSTITUTE
University of California, Los Angeles

Working Paper No. 362

September, 1988

REUSING STRUCTURED MODELS VIA MODEL INTEGRATION

by

Arthur M. Geoffrion

Abstract

This paper begins with a review of reusability and modularity ideas from the software engineering literature, most of which are applicable to the modeling context. Many features of structured modeling support reusability and modularity, and these are noted. The main focus of the paper, however, is on achieving reusability and modularity via the integration of two or more model schemas. A 5-step approach for integrating schemas written in SML (Structured Modeling Language) is proposed for this purpose. Examples are given to illustrate this approach, and the pros and cons of structured modeling for reuse are discussed at some length. (KR) ←

Accepted for publication in *Proceedings of the Twenty-Second Hawaii International Conference on System Sciences*, to be held January, 1989 in Kailua-Kona, Hawaii.

Acknowledgments

I gratefully acknowledge the helpful comments of Gordon Bradley, Dan Dolk, Sergio Maturana, Laurel Neustadter, Yao-Chuan Tsai, and Fernando Vicuña.

Supported partially by the National Science Foundation, the Office of Naval Research, and the Navy Personnel Research and Development Center (San Diego). The views contained in this report are those of the author and not of the sponsoring agencies.

REUSING STRUCTURED MODELS VIA MODEL INTEGRATION

"Reusability" and "modularization" concepts play a key role in the part of software engineering that is concerned with improving software development productivity. The analogies between programming and modeling suggest that some of what software engineers have learned about these concepts might be adapted to improve model development productivity.

Reusability need not involve model integration, but integration is an important way in which reuse can occur in the context of modeling. Reuse via model integration is the central focus of this paper.

In particular, we treat this topic from the viewpoint of *structured modeling* (Geoffrion <1987a> <1987c>). It would be impossible to give an adequate introduction to structured modeling within the confines of this paper, and so we must presume prior knowledge on the part of the reader. In a nutshell, however, one can say that structured modeling aims to provide a formal conceptual framework of considerable generality for modeling, within the broader task of laying the foundation for a new generation of modeling environments. The framework uses a hierarchically organized, partitioned, and attributed acyclic graph to represent the semantic as well as mathematical structure of a model. A language called SML has been designed and implemented to support this framework (Geoffrion <1988a>).

The organization of this paper as follows. Section 1 reviews reusability and modularity ideas from the software engineering literature for the benefit of readers who may not be familiar with these ideas. Section 2 presents a procedure for integrating the general structure of two distinct models written in SML. Section 3 and a series of examples in Appendices 1-3 illustrate this approach in simple but instructive settings. The last section discusses some of the issues that arise when one considers structured modeling in the context of the ideas presented in the prior sections.

Papers by other authors on model integration include Blanning <1986>, Bradley and Clemence <1988>, Kottemann and Dolk <1988>, Liang <1986>, Muhanna and Pick <1988>, Murphy et. al. <1988>, Tsai <1987>, and Zeigler <1984> (Chaps. 4,5). Each adds a dimension not treated here. For example, Kottemann and Dolk <1988> addresses the integration of model manipulation operations, whereas we limit consideration to model definition alone. For a more general discussion of model integration, see Geoffrion <1987b>. That paper distinguishes several different kinds of model integration and assesses the strengths and weaknesses of structured modeling in support of each.

1. SELECTED IDEAS FROM SOFTWARE ENGINEERING

Reusability is a prominent topic in software engineering because being able to use previously created and (presumably) debugged code saves work that is expensive and for which skilled personnel are in short supply. Advantages accrue not only to the initial development activity, but also to the very important activities of maintenance and modification. Moreover, code is not the only thing that can be reused; it can be advantageous also to reuse data, program and system designs and architecture, specifications, test plans, and so on. See Freeman <1987> for an excellent collection of readings on reusability.

As might be expected, different authors focus on different aspects of reusability. Here is a selected sample of points and opinions.

One recent article by a leading expert on software engineering cites "reusing components" as one of six primary options for improving software productivity (Boehm <1987>). To Boehm, reusability is most closely related to two things: libraries of software components (like mathematical and statistical routines) and 4GL/application generators (like Focus and Nomad). The justification for the second item appears to be that various of the software components constituting a 4GL processor or application generator are reused at each application in place of writing a program in a traditional language. Reuse is by reconfiguring the pre-existing components into a relatively special purpose software system, or by using them in lieu of a special purpose software system by "executing" a high level specification of functionality.

Near the end of the article, Boehm offers this opinion:

> "Thus, for the 1995-2000 time frame, we can see that two major classes of opportunities for improving software productivity exist: providing better support systems for broad-domain applications, involving fully integrated methods, environments, and modern programming languages such as Ada; and extending the number and size of the domains for which we can use domain-specific fourth-generation languages and application generators."

If transposed from the context of programming to the context of modeling, this view is very much in the spirit of structured modeling.

Another view of reusability is provided by Balzer, Cheatham, and Green <1983>. Their basic approach is that software should be produced by end users who write formal specifications that can be transformed more or less automatically by computer into the desired software. Maintenance would not be done on source code, as is the usual case today, but rather on the formal specification; revised code would be regenerated as needed. This automation-based paradigm for software development admittedly is idealistic: the necessary technology is not yet adequate for many applications. Thus the authors accept skilled intervention with respect to strategic implementation decisions. However, they argue that both the documentation of these decisions (they call this the "development") and their execution should be automated. The authors claim important advantages for the automation-based paradigm, among which is software reusability. To quote:

> "Software libraries - with the exception of mathematical subroutine libraries - have failed because the wrong things are in those libraries. They are filled with implementations, which necessarily contain many arbitrary decisions. The chance that such an implementation is precisely right for some later application is exceedingly small.
>
> Instead, we should be placing specifications and their recorded development in libraries. Then, when a module is to be reused, the specification can be modified appropriately - that is, maintained - and its development changed accordingly. Thus, maintenance of both the specification and its development becomes the basis of reuse, rather than a fortuitous exact match with a previously stored implementation."

There seems to be a strong connection between the automation-based paradigm and the 4GL/application generators advocated by Boehm: the latter can be viewed as domain-specific examples of the former.

A third article discussing reusability, Kartashev and Kartashev <1986>, takes the position that the reuse of software components is one of only two main options for reducing software costs. Within this option, two approaches are noted:

> "... The *composition approach* provides for some abstract initial representation of reused program modules, then the use of a composition technique such as parameterized programming, the UNIX pipe mechanism, or special specification techniques to construct a complex program from reusable modules with the interfaces specified by the composition technique.
>
> The second approach involves the creation of libraries of reused components that do not need preliminary specifications to be connected to each other."

Goguen <1986> develops the first approach in detail. He gives this brief and tantalizing synopsis at the outset:

> "...The goal is to make programming significantly easier, more reliable, and cost effective by *reusing* previous code and programming experience to the greatest extent possible. Suggestions given here include: systematic (but limited) use

of semantics, by explicitly attaching *theories* (which give semantics, either formal or informal) to software components with *views* (which describe semantically correct interconnections at component interfaces); use of *generic* entities, to maximize reusability; a distinction between *horizontal* and *vertical* composition; use of a *library interconnection language*, called LIL, to assemble large programs from existing entities; support for different levels of formality in both documentation and validation; and facilitation of program understanding by animating abstract data types and module interfaces."

Several of these ideas have analogues in structured modeling.

Goguen's most intriguing ideas, and his main focus, are those relating to LIL. The structured modeling analogue of LIL would be a language for integrating models (schemas or even fully specified structured models). No such language presently exists for structured modeling. In the modeling literature more generally, the only efforts I know of along these lines are the recent contributions by Bradley and Clemence <1988> and by Muhanna and Pick <1988>. Since such a language could greatly facilitate model reuse through integration, this is a potentially fruitful area for research. Some inspiration may be found in the literature on module interconnection languages, which has been surveyed by Prieto-Diaz and Neighbors <1986>.

A topic in software engineering closely related to reusability is *modularization*. The connection is that well modularized code is easier to reuse than code that is poorly modularized. Presumably, the same statement is true of models. See, for example, the classic paper by Parnas <1972>, which cites these benefits for a modular style of programming:

* <u>shorter development time</u> is possible because different people can work simultaneously on separate program modules;

* <u>evolutionary flexibility</u> is improved because drastic changes can be made to one module without impacting the others;

* <u>understandability</u> is improved because it should be possible to study the system one module at a time.

That paper champions <u>information hiding</u> as a criterion that often leads to better modularization than more traditional criteria. That is, each module "hides some design decision from the rest of the system", with its interfaces crafted "to reveal as little as possible about its inner working." The design decisions worth modularizing are those which are difficult or likely to change.

Modularization ideas play a major role in the thinking of many leading authors in the area of program and system design. For example, Yourdon and Constantine <1979> describe the central focus of their book as follows:

"Our concern is with the architecture of programs and systems. How should a large system be broken into modules? *Which* modules? Which ones should be subordinate to which? How do we know when we have a 'good' choice of modules? and, more important, How do we know when we have a 'bad' one? What information should be passed between modules? Should a module be able to access data other than that which it needs to know in order to accomplish its task? How should the modules be 'packaged' into efficient executable units in a typical computer?" [page xvi]

Much of the book's discussion of such questions can be translated easily into provocative positions about analogous questions in modeling. These positions deserve to be studied.

One of the most fashionable manifestations of modularization and related ideas in the service of reusability (and other popular goals) is *object-oriented programming*. A particularly vigorous presentation from this point of view is found in the book by Cox <1987>. He argues that it is possible to develop software counterparts of integrated circuits; counterparts in that they can be reused readily by programmers much as integrated circuits are reused readily by hardware designers. Here are two intriguing passages to provide a taste of Cox's viewpoint:

"Objects are tightly encapsulated and thus relatively independent of their environment. They can be designed, developed, tested, and documented as stand-alone units without knowledge of any particular application, and then stockpiled for distribution and reuse in many different circumstances. A reusable module acquires value unto itself, while the value of application dependent code has no value beyond that of the application as a whole. It can be tested and documented to much higher standards than has been done in the past." [page 26]

"These concepts, working together, put reusability at the center of the programming process. Encapsulation raises firewalls around the objects within a system, so that if they change, other parts of the system can remain unchanged. Inheritance does the converse, by spreading decisions implemented in generic classes automatically to each of their subclasses. The net effect is making libraries of reusable code, called Software-IC libraries, a central feature of the system-building shop." [page 91]

This completes the review of reusability and modularization ideas from the software engineering literature.

We turn now to a step-by-step approach for integrating two structured modeling schemas. This is one way in which reuse can occur in modeling. Each participating schema can be viewed as a module.

2. A PROCEDURE FOR MODEL INTEGRATION

This section explains a procedure for integrating the general structure of two models. Let two SML model schemas be given.

By necessity, we presume that the reader is acquainted with the basic technical terms of SML (Geoffrion <1988a>). For example, the reader should know that an SML *schema* represents a model's general structure; that *Schema Properties* are context-sensitive syntax rules associated with the schema; and that it falls to SML's *elemental detail tables* to instantiate a schema to a specific model instance. In what follows, a "structural change" is any change other than simple renaming or changing an interpretation.

Step 1. Customize the two schemas to the roles that they will play in the integrated model, but do not violate the syntax or Schema Properties of SML. For example, achieve consistency in units of measurement.

a. Make any appropriate structural changes.

b. Customize the genus names, module names (including the root module name), key phrases, and interpretations of each schema as appropriate. Often this helps to reduce the amount of work that must be done at Step 3a.

Step 2. Make technical preparations for joining the two schemas, but do not violate the syntax or Schema Properties of SML.

a. Anticipate duplicate genus names, module names, key phrases, indices, functional and multi-valued dependency names, and symbolic parameter stems, that will occur at Step 3, and eliminate all conflicts.

b. Usually there is at least one pair of genera, one from each schema, that needs to be "merged". Identify all such pairs. Make ready for the merges by making both genus paragraphs as similar as possible for each merging genus pair. Sometimes they can be made identical, but not always. Structural changes are allowed.

c. Optionally, put the "inputs" and "outputs" of each schema in special modules. This may not be possible to do in a fully satisfactory way. Structural changes are allowed.

Step 3. Join the two schemas. This step can cause Schema Property violations.

a. Concatenate the two schemas. Choose an order that preserves monotonicity of the modular structure if possible.

b. Continue the work begun in Step 2b, if necessary, until each merging genus pair is identical. Drop the second genus replicate for each merging genus pair.

c. Make any other structural changes necessary for proper joining.

Step 4. Revise the integrated schema as necessary, structurally or non-structurally, to restore satisfaction of all Schema Properties. For example, it may be necessary to revise the modular structure so as to restore monotonicity.

Step 5. Optionally, make any desired final cosmetic or structural changes, being sure to observe the syntax and Schema Properties of SML.

The next section presents an illustrative application of this approach. Actually, the approach is carried out three times in a cumulative way that culminates in the integration of four distinct models.

Additional illustrative applications are given in the first three appendices. They apply the approach to three simple situations. The first integrates a forecasting model with the classical transportation model. The second integrates the transportation model with a multi-item economic order quantity model. The third integrates two transportation models to produce a two-echelon transshipment model.

All examples treat model schemas, not specific model instances. It would be worth examining these examples at the level of fully detailed instances, but that is not done here.

3. AN EXAMPLE

This section illustrates the approach of Section 2 with a small but richly instructive example that has been used by several authors. The example is due to Blanning, who discusses model integration at some length in several of his papers (e.g., Blanning <1986>, Blanning <1987>). A recurring example in his papers is the "CORP" model, which has been discussed also by other authors (e.g., Muhanna <1987>).

One of the interesting things about this model is that it can be viewed as four interconnected submodels. The development of this section explains how the appropriate interconnections can be accomplished by applying the integration procedure of Section 2 to SML representations of the component submodels.

The CORP model is discussed at length in Geoffrion <1988b> because although it appears to be cyclic, closer inspection reveals that it need not be. The Third Formulation therein repairs certain anomalies of the original model and presents it as a genuine SML schema. That is the version used here. It can be considered to be what the final integrated model should look like, except perhaps for its overly simple modular structure.

Original Component Models

There are four component models. The first estimates demand.

&MKT DEMAND ESTIMATION

PROD /pe/ There is a certain PRODUCT.

P (PROD) /a/ : Real+ The PRODUCT has a unit PRICE.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales VOLUME given by a certain linear demand function in terms of PRICE.

The second model analyzes markup. It is the only component model that differs significantly from the corresponding one of Blanning. It does not calculate P from the values of **M**, V, and E. However, that value of P can be calculated by a solver for the satisfaction problem of finding P such that **M_VAR** is zero.

&MAR MARKUP ANALYSIS

PROD /pe/ There is a certain PRODUCT.

P (PROD) /va/ : Real+ The PRODUCT has a unit PRICE.

V (PROD) /a/ : Real+ The PRODUCT has a sales VOLUME.

E (PROD) /a/ : Real+ There is a total MANUFACTURING EXPENSE associated with the PRODUCT.

M (P,V,E) /f/ ; P * V / E The actual MARKUP for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the MARKUP VARIANCE.

The third component model predicts manufacturing cost.

&MFG MANUFACTURING

PROD /pe/ There is a certain PRODUCT.

U (PROD) /a/ : Real+ The PRODUCT has a UNIT COST of manufacture, exclusive of fixed manufacturing expenses.

V (PROD) /a/ : Real+ The PRODUCT has a sales VOLUME.

E (U,V) /f/ ; 1000000 + U * V The total MANUFACTURING EXPENSE for the PRODUCT is fixed manufacturing expenses plus UNIT COST times VOLUME.

The fourth model one calculates a key financial quantity.

&FIN FINANCIAL

PROD /pe/ There is a certain PRODUCT.

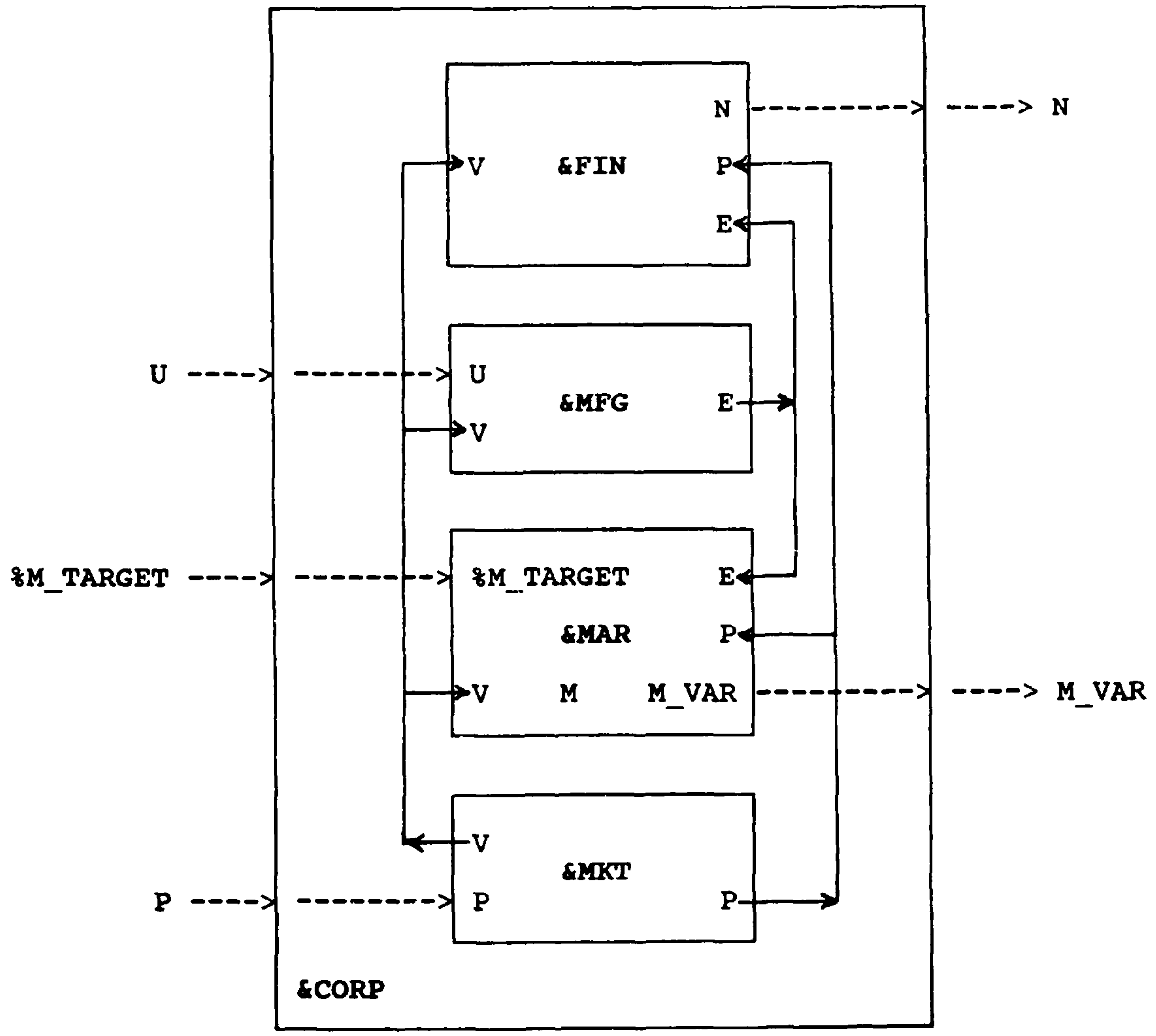
P (PROD) /a/ : Real+ The PRODUCT has a unit PRICE.

V (PROD) /a/ : Real+ The PRODUCT has a sales VOLUME.

E (PROD) /a/ : Real+ There is a total MANUFACTURING EXPENSE associated with the PRODUCT.

N (P,V,E) /f/ ; P * V - E The NET INCOME obtained for the PRODUCT is total revenue (PRICE times VOLUME) less total MANUFACTURING EXPENSE.

It is straightforward to interconnect these four components into a single composite model provided there is sufficient compatibility among definitions that appear in more than one component model. This is a critical requirement whenever component models are to be interconnected. Care has been taken to ensure that it holds here. The following interconnection diagram is adapted from Muhanna and Pick <1986>.



Note that **&MKT** has the task of supplying the value of **V** to the other three models, **&MFG** the value of **E** to two of the other models, **&FIN** the value of **N**, and **&MAR** the value of **M_VAR**. All these are obvious roles because **V**, **E**, **N**, and **M_VAR** are computed only once in the four models. **U** is not computed at all, and so must be supplied externally. The same is true for the symbolic parameter %M_TARGET. In addition, **&MKT** passes on the externally supplied value of **P**, which is modeled as a variable attribute, to the two other models that need it. This is one of two significant differences between the above diagram and the Muhanna-Pick diagram on which it is based; their diagram would receive and distribute **P** from **&MAR** (which they call "PRI") instead of from **&MKT**. If our diagram did it this way, it would be cyclic rather than acyclic. The second significant difference is noted in the next paragraph.

Thus **&CORP** can be viewed as having three inputs, **U**, %M_TARGET, and **P**, and two outputs, **N** and **M_VAR**. If a value of **P** is desired such that **M_VAR** is zero, then finding such a value is the task of a solver external to **&CORP**. Please note that this viewpoint differs from that of Blanning and Muhanna, who both appear to view **&CORP** itself (as a model) as bearing the responsibility for finding such a value. I believe that such a viewpoint confuses the notion of model and solver.

There is nothing to prevent the four component models from being interconnected per the diagram by some mechanism <u>external</u> to the SML notational system. Perhaps a capability of this sort should be provided by a modeling environment based on structured modeling; a formal interconnection language would be one possibility. If so, however, some defenses are needed to detect and repair definitional incompatibilities when similar model elements appear in multiple component models. (This is not unlike the problems that arise in information systems when distinct databases are to be integrated with one another; see, e.g., Batini, Lenzerini, and Navathe <1986>).

That is not the approach taken here. In what follows, we integrate the four models by stages into a single composite SML schema. First, **&MKT** and **&MAR** will be integrated. Then **&MFG** will be folded in. Finally, **&FIN** will be integrated with the result of the previous two integrations.

It is evident from the partial ordering of the component models (such an ordering is obvious from the interconnection diagram) that it would be wiser to interchange the integration order of **&MAR** and **&MFG**, for the former depends on the latter. Wiser in the sense that monotonicity would be easier to maintain. We choose the less favorable order to help dispel the reader's possible suspicion that choosing the order of integration requires divine guidance.

Integrate &MKT and &MAR

Step 1. Null.

Step 2a. Null.

Step 2b. The merging genus pairs are PROD, P, and V in &MKT and, respectively, PROD, P, and V in &MAR. The PROD pair already has identical genus paragraphs. The P paragraphs differ only in that the first is /a/ and the second is /va/; make the first identical to the second. The V paragraphs differ substantially; make the second identical to the first (since the second paragraph will be dropped at Step 3b, it is not truly necessary to make this change).

Step 2c. Omit.

Step 3a. Concatenate in the order &MKT, &MAR. Call the new module &MKT_MAR.

Step 3b. Drop the PROD, P, and V paragraphs from &MAR.

Step 3c. Null.

Step 4. Null.

Step 5. Omit.

The result is as follows.

&MKT_MAR

&MKT <u>DEMAND ESTIMATION</u>

PROD /pe/ There is a certain <u>PRODUCT</u>.

P (PROD) /va/ : Real+ The PRODUCT has a unit <u>PRICE</u>.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales <u>VOLUME</u> given by a certain linear demand function in terms of PRICE.

&MAR <u>MARKUP ANALYSIS</u>

E (PROD) /a/ : Real+ There is a total <u>MANUFACTURING EXPENSE</u> associated with the PRODUCT.

M (P,V,E) /f/ ; P * V / E The actual <u>MARKUP</u> for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the <u>MARKUP VARIANCE</u>.

Integrate &MKT_MAR and &MFG

Step 1. Null.

Step 2a. Null.

Step 2b. The merging genus pairs are PROD, V, and E in &MKT_MAR and, respectively, PROD, V, and E in &MFG. The PROD paragraphs already are identical. The V paragraphs differ substantially; the second needs to be made like the first, but this cannot be accomplished until Step 3b because &MFG has no P genus. The E paragraphs also differ substantially; the first needs to be made like the second, but this cannot be accomplished until Step 3b because &MKT_MAR has no U genus. Thus no changes at all are made at this substep.

Step 2c. Omit.

Step 3a. Concatenate in the order &MKT_MAR, &MFG. Call the new module &MKT_MAR_MFG.

Step 3b. Drop the second PROD paragraph. Make the second V paragraph identical to the first, then drop the second (obviously, it is not truly necessary to change V before dropping it). Make the first E paragraph identical to the second, then drop the second one.

Step 3c. Null.

Step 4. There is only one Schema Property violation, the one caused by the forward reference to U by the E paragraph. Cure it by moving the &MFG module to the position between &MKT and &MAR.

Step 5. Move the E paragraph to the position immediately after the U paragraph. This is more in keeping with the scopes of &MAR and &MFG. Change the modular structure (here we do not bother to list the genera)

	&MKT_MAR_MFG		&MKT_MFG_MAR
	&MKT_MAR		&MKT
from	&MKT	to	&MFG
	&MFG		&MAR.
	&MAR		

The result is as follows.

&MKT_MFG_MAR

&MKT DEMAND ESTIMATION

PROD /pe/ There is a certain PRODUCT.

P (PROD) /va/ : Real+ The PRODUCT has a unit PRICE.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales VOLUME given by a certain linear demand function in terms of PRICE.

&MFG MANUFACTURING

U (PROD) /a/ : Real+ The PRODUCT has a UNIT COST of manufacture, exclusive of fixed manufacturing expenses.

E (U,V) /f/ ; 1000000 + U * V The total MANUFACTURING EXPENSE for the PRODUCT is fixed manufacturing expenses plus UNIT COST times VOLUME.

&MAR MARKUP ANALYSIS

M (P,V,E) /f/ ; P * V / E The actual MARKUP for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the MARKUP VARIANCE.

Integrate &MKT_MFG_MAR and &FIN

Step 1. Null.

Step 2a. Null.

Step 2b. The merging genus pairs are PROD, P, V, and E in &MKT_MFG_MAR and, respectively, PROD, P, V, and E in &FIN. The PROD and P pairs already have identical genus paragraphs. The V paragraphs differ substantially; make the second identical to the first (since the second paragraph will be dropped at Step 3b, it is not truly necessary to make this change). The E paragraphs also differ substantially; the second needs to be made like the first, but this cannot be accomplished until Step 3b because &FIN has no U genus.

Step 2c. Omit.

Step 3a. Concatenate in the order &MKT_MFG_MAR, &FIN. Call the new module &CORP.

Step 3b. Drop the second PROD and P paragraphs. Drop the second V paragraph. Make the second E paragraph identical to the first, then drop the second one (obviously, it is not truly necessary to change E before dropping it).

Step 3c. Null.

Step 4. Null.

Step 5. Change the modular structure

```
           &CORP                             &CORP
              &MKT_MFG_MAR                      &MKT
from             &MKT                to         &MFG
                 &MFG                           &MAR
                 &MAR                           &FIN.
              &FIN
```

The result is as follows. Clearly it is consistent with the interconnection diagram given earlier. In fact, the diagram gives an accurate view of &CORP.

&CORP

&MKT DEMAND ESTIMATION

PROD /pe/ There is a certain PRODUCT.

P (PROD) /va/ : Real+ The PRODUCT has a unit PRICE.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales VOLUME given by a certain linear demand function in terms of PRICE.

&MFG MANUFACTURING

U (PROD) /a/ : Real+ The PRODUCT has a UNIT COST of manufacture, exclusive of fixed manufacturing expenses.

E (U,V) /f/ ; 1000000 + U * V The total MANUFACTURING EXPENSE for the PRODUCT is fixed manufacturing expenses plus UNIT COST times VOLUME.

&MAR MARKUP ANALYSIS

M (P,V,E) /f/ ; P * V / E The actual MARKUP for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the MARKUP VARIANCE.

&FIN FINANCIAL

N (P,V,E) /f/ ; P * V - E The NET INCOME obtained for the PRODUCT is total revenue (PRICE times VOLUME) less total MANUFACTURING EXPENSE.

Modular structure aside, &CORP is indeed exactly the same as the Third Formulation in Geoffrion <1988b>, as predicted at the outset of this section.

This completes our application, thrice in succession, of the SML schema integration approach given in Section 2.

4. DISCUSSION

This section discusses reusability and related ideas sketched in Section 1 from the point of view of structured modeling and the examples of model integration provided in Section 3 and the appendices.

ISSUE: How well does structured modeling support "information hiding" and "modularization"?

Several features of structured modeling promote the objectives of information hiding and modularization:

(1) the core concept of hierarchical *modular structure* (Definition 12 of Geoffrion <1987c>);

(2) the concepts of *views* and *modular outlines* (Definitions 20 and 23 of Geoffrion <1987c>), and the closely related notion of using outline processor functions to conceal and reveal different portions of modular structure depending on the intended audience (Section 4.3 of Geoffrion <1987a>);

(3) the sharp distinction between models and solvers (Section 2.3 of Geoffrion <1987a>) helps to keep the internal concerns of each away from the other;

(4) model and solver libraries (Sections 2.4 and 4.3 of Geoffrion <1987a>) achieve a kind of information hiding and modularization at the macroscopic level (although some of the models in a model library could be relatively small reusable fragments);

(5) the distinction in SML between the general structure of the schema and the detailed data of the elemental detail tables enables users to be insulated from an excessive volume of problem specifics when that is appropriate;

(6) the distinction in SML between the formal part and the interpretation of each genus and module paragraph helps to insulate technical users from excessive problem domain information, and non-technical users from excessive technical details; and

(7) each individual element in a structured model is, in effect, a module (e.g., the details of the rule of a function element are totally localized).

These features may not go as far toward the support of information hiding and modularization as some people might wish (cf. Muhanna and Pick <1986>). Certainly they do not go as far as the notion of a "black box," taken here to mean a hiding or module so secure that special authority is required before one is permitted to look inside it. Black boxes don't make sense in the context of modeling if integration is to be an important kind of reuse. The reason is that almost any part of a model can serve as the focus of integration with another model. Since integration is difficult or impossible if the focal part is inaccessibly hidden, it follows that virtually no part of a model can be black boxed without limiting the possibilities for integration. Obviously, to limit the possibilities for integration is to limit the possibilities for reuse.

To illustrate that almost any part of a model can serve as the focus of integration, consider the case of the classical transportation model. The development of Appendix 1 illustrates that the demand portion can be the focus of integration (with a forecasting model). The development of Appendix 2 illustrates that the transportation links can be the focus of integration (with an economic order quantity model). The development of Appendix 3 illustrates that the origins and their supply capacities can be the focus of integration (with another transportation model on the front end), and that the destinations and their demands can be the focus of integration (with another transportation model on the back end). It would be easy to develop other examples in which the transportation cost rate is the focus of integration with a freight rate estimation model, or in which the supply capacities are integrated with a more detailed supply capacity estimation model. Thus every part of the classical transportation model can serve as the focus of integration.

The most extreme form of information hiding is close to the idea of a "black box." How does one know whether a black box contains something that is truly consistent with what it is connected to? This is a particularly difficult worry in the realm of modeling, where model elements often have subtle definitions. The only way to know for sure is for the modeler to inspect the contents of the box. The contents alone suffice to explain the complete function of the box. **Unit of measurement?** Look inside to find out whether it is pounds, hundredweight, tons, or some other unit. **Demand point?** Look inside to find out whether it is an individual customer, all customers of a given class of trade in a certain area, etc. **Product?** Look inside to find out whether tan and black products are grouped, whether standard attachments are included with the basic product itself, and so on. **Cost?** Only a look inside will tell you whether it is book, actual, incremental, fully or partially loaded, etc. It would be grossly redundant to attach all this information to the "inputs" and "outputs" of a black boxed model, which one would have to do to accommodate all possible kinds of reuse.

This establishes that information hiding and modularization ought not to be taken to the black box extreme in the context of modeling. However, I accept these as criteria to help guide the practical use of features (1)-(4) listed above.

It also establishes that visibility of all model parts and explicit representation of their interconnections are important objectives in support of reuse through integration.

Structured modeling comes close to making a fetish of these objectives. For example, the core concept of a *calling sequence* (Definition 6 of Geoffrion <1987c>) renders interconnections explicit, and the interpretation part of genus and module paragraphs in SML help to make all model parts visible to users who may find difficulty in reading the more formalistic parts of SML. "Input" modules or interfaces are not necessary because calling sequences in effect play this role. The generic calling sequence in SML does the same thing at the level of an entire genus. Nor are "output" modules or interfaces necessary in structured modeling, because any element or genus can be called by any other (subsequent) one that needs it as input.

The above comments have to do with how structured modeling relates to some of the ideas reviewed in Section 1. What about the model integration approach of Section 2? A brief answer is that Step 2c can increase the degree of modularization, but that Step 3b tends to decrease it by increasing the level of structural connectivity. It is not clear whether the latter effect is good or bad on balance, as its negative effects on maintainability and reusability are offset by positive effects on efficiency and avoidance of redundancy.

ISSUE**: To what extent can the 5-step approach of Section 2 be automated?**

The approach detailed in Section 2 and illustrated in Section 3 and Appendices 1-3 evidently is quite labor intensive. Certain standard capabilities of ordinary word processors, like the search/replace and block move features, have obvious uses for carrying out many of the substeps, but a higher degree of automation seems essential to facilitate model integration in general.

A tree-oriented editor, otherwise known as an "outliner", would be helpful. (We note that one is available in FW/SM, the prototype structured modeling implementation.) It would be much more helpful to have a SML syntax-directed (structure) editor along the lines proposed by Vicuña <1988>: it could detect or even preclude the violations of SML syntax or Schema Properties prohibited in Steps 1, 2, and 5, and could detect the violations that must be fixed at Step 4.

The diagnostic part of Step 2a lends itself particularly well to automation, as does -- although it is more difficult -- the diagnostic part of Step 4. Moreover, specialized search-and-replace capabilities for changing names, indices, and key phrases would be helpful for Steps 1b, 2a, and perhaps other steps.

A good implementation of all or even most of these possibilities would greatly facilitate the integration of structured model schemas. However, there will remain a certain amount of modeler's discretion that must be exercised at most of the steps. For example, only the modeler can decide at Step 1a exactly what structural changes are appropriate. Similar discretion is needed at the other steps, except for 3a, 2a (probably), and 3b (possibly). That discretion, incidentally, is what an integration language like that of Bradley and Clemence <1988> must express.

The last-mentioned reference, and Murphy, Stohr, and Ma <1988>, are examples of recent work that offer intriguing prospects for automating some of what we have called "modeler's discretion" by more deeply exploiting the syntactic and semantic information available in (or attachable to) two SML schemas.

ISSUE: Within the context of SML, is it easier to reuse schemas per the integration approach of Section 2, or to build monolithic schemas from scratch?

I believe that reuse will be significantly more efficient than ground-up development in most cases. The amount of work that must be done with each approach in a particular application depends on the special facilities available in the modeling environment and the answer to the previous issue.

It should be possible to address this issue experimentally by counting operations (or actions, or deliberate decisions that must be made, or the amount of time consumed) for each approach for different modelers in a controlled setting. This is an attractive research topic.

There may be some modifications to SML that would reduce the total amount of work that must be done in reuse via model integration. For example, it can be a nuisance to have to avoid duplication of names for genera, modules, indices, functional and multi-valued dependencies, and symbolic parameter stems. It might be feasible to allow selected modules to limit the scope of the names defined within them, so that all external references would have to include the name of the defining module in order to gain access. Duplicate names could then be allowed because references could be unambiguous. Such an extension of SML would be well worth working out.

ISSUE: Within the context of FW/SM, the prototype structured modeling implementation, does structured modeling put the "right" things put in libraries in the sense of Balzer, Cheatham, and Green <1983>?

Structured modeling puts model schemas, elemental detail tables, and solvers in libraries.

A schema is an analog of a "specification" in the sense of Balzer, Cheatham, and Green <1983>. Putting a schema in a library seems appropriate to the extent that FW/SM facilitates revising it and integrating it with other schemas, provided that it is of possible future interest. See Sections 3.1 and 3.2 of Geoffrion <1987a> and the appendices of this paper.

Putting not only a schema, but also associated elemental detail, in a library makes as much sense as putting just the schema in the library in some situations. One such occurs when the elemental detail is a commercial or private database with multiple potential applications, such as an air freight rate database that is of possible use in any logistics model involving air freight. Another such situation occurs when the aim is to set the stage for model integration within the context of the original application. Of course, there could not be many more than one reuse of elemental detail in this case.

There can be little question about the appropriateness of putting solvers in libraries, as the function they perform is useful over the entire class of models to which they are intended to apply.

While on the subject of libraries, it should be noted that a library is of little use if users cannot easily find out what is in it. This suggests that a structured model of the contents would be useful. More generally, it could be useful to have computer-based tools to support library management and access. One paper dealing specifically with libraries in the context of model management is Mannino, Greenberg, and Hong <1987>. Related papers in the context of software, such as Prieto-Diaz and Freeman <1987>, may be inspirational.

ISSUE: In programming, the goals of reusability, maintainability, and modifiability usually are pursued by adhering to strict disciplines like structured programming, structured design, etc. that impose order on the otherwise chaotic process of programming. Similar goals for modeling would seem to imply the need for comparably strict disciplines that impose order on the otherwise chaotic process of modeling. Does structured modeling impose such discipline?

Yes, structured modeling does indeed impose a great deal of discipline on the modeler by comparison with what would be the case if, say, ordinary mathematics were used in the usual *ad hoc* fashion to express models. Consider:

* All definitional dependencies among model elements are supposed to be explicit (in the calling sequences).

* Circular definitional dependencies are supposed to be avoided.

* All values are supposed to have an explicit range.

* Modular structure is supposed to reflect the natural conceptual groupings of model elements.

* Model elements are supposed to be arranged in a sequence such that there are no forward definitional references.

* Model instances are supposed to be viewed in terms of a general class of model instances that are all isomorphic in a certain sense.

* Models are supposed to be sharply distinguished from problems posed on models, and both of these from solvers.

These conventions are imposed by the core concepts of structured modeling (Geoffrion <1987c>). SML either enforces or is consistent with all of these conventions, and lays down others as well (e.g., the strict separation of general structure from detailed data, and the provision for informal interpretation as an integral part of formal specification).

The overall intended effect of these and other conventions is to bring order to the modeling process in a way that will advance the goals of reusability, maintainability, and modifiability.

REFERENCES

BALZER, R., T.E. CHEATHAM, Jr., and C. GREEN <1983>. "Software Technology in the 1990's: Using a New Paradigm," *Computer*, 16:11 (November), pp. 39-45.

BATINI,C., M. LENZERINI, and S.B. NAVATHE <1986>. "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, 18:4 (December), pp. 323-364.

BLANNING, R.W. <1986>. "A Relational Framework for Information Management," in E.R. McLean and H.G. Sol (eds.), *Decision Support Systems: A Decade in Perspective*, Elsevier Science Publishers B.V. (North-Holland), Amsterdam.

BLANNING, R.W. <1987>. "A Relational Theory of Model Management," in C.W. Holsapple and A.B. Whinston (eds.), *Decision Support Systems: Theory and Application*, Springer-Verlag.

BOEHM, B.W. <1987>. "Improving Software Productivity," *Computer*, 20:9 (September), pp. 43-57.

BRADLEY, G.H. and R.D. CLEMENCE, Jr. <1988>. "Model Integration with a Typed Executable Modeling Language," *Proceedings of the Twenty-First Hawaii International Conference on System Sciences, Vol. III*, January, pp. 403-410.

COX, B.J., Jr. <1987>. *Object-Oriented Programming*, Addison-Wesley, Reading, MA.

FREEMAN, P. <1987>. *Software Reusability*, IEEE Cat. No. EH0256-8, Computer Society Press, Washington, D.C.

GEOFFRION, A. <1987a>. "An Introduction to Structured Modeling," *Management Science*, 33:5 (May), pp. 547-588. A version that includes a section on implementation will also appear in *Proceedings of the Conference on Integrated Modeling Systems* (held at the University of Texas, Austin, October 1986). The latter version is available as Working Paper No. 338, Western Management Science Institute, UCLA, 75 pages, 6/86. Last revised March, 1988. This is the version referenced here.

GEOFFRION, A. <1987b>. "Integrated Modeling Systems," Working Paper 343, Western Management Science Institute, UCLA, 16 pages, 11/86. Last revised March, 1988. To appear in the *Proceedings of the Conference on Integrated Modeling Systems* (held at the University of Texas, Austin, October 1986).

GEOFFRION, A. <1987c>. "The Formal Aspects of Structured Modeling," forthcoming in *Operations Research*. Available as Working Paper 346, Western Management Science Institute, UCLA, 39 pages, 5/87. Last revised August, 1988.

GEOFFRION, A. <1988a>. "SML: A Model Definition Language for Structured Modeling," Working Paper 360, Western Management Science Institute, UCLA, 129 pages, May.

GEOFFRION, A. <1988b>. "A 'Cyclic' Model Discussed by Blanning and Muhanna," Informal Note, UCLA, 10 pages, June 5. Revised August 4.

GEOFFRION, A. <1988c>. "A Library of Structured Models," Informal Note, UCLA, 200 pages, 6/30/88.

GOGUEN, J.A. <1986>. "Reusing and Interconnecting Software Components," *Computer*, 19:2 (February), pp. 16-28. Reprinted in Freeman <1987>.

KARTASHEV, S. and S. KARTASHEV <1986>. "Guest Editors' Introduction," *Computer*, 19:2 (February), pp. 9-13.

KOTTEMANN, J.E. and D.R. DOLK <1988>. "Process-Oriented Constructs for Model Integration," working paper, Naval Postgraduate School, 28 pages, March. Submitted for publication.

LIANG, T.P. <1986>. *Toward the Development of a Knowledge-Based Model Management System*, Ph.D. Dissertation, University of Pennsylvania.

MANNINO, M., B.S. GREENBERG and S.N. HONG <1987>. "Knowledge Representation for Model Libraries," Working Paper CBDA 143, Center for Business Decision Analysis, the University of Texas at Austin, August.

MUHANNA, W. <1987>. "Composite Models: Hierarchical Construction, Circularity, and Deadlocks," Working Paper, University of Wisconsin - Madison, October.

MUHANNA, W.A. and R.A. PICK <1986>. "A Systems Framework for Model Management," School of Business, Univ. of Wisconsin-Madison, December. Revised August, 1988.

MUHANNA, W.A. and R.A. PICK <1988>. "Composite Models in SYMMS," *Proceedings of the Twenty-First Hawaii International Conference on System Sciences, Vol. III*, January, pp. 418-427.

MURPHY, F.H., E.A. STOHR and P. MA <1988>. "Composition Rules for Building Linear Programming Models from Component Models," Working Paper, Information Systems Area, Graduate School of Business Administration, New York University, June 17.

PARNAS, D.L. <1972>. "On the Criteria To Be Used in Decomposing Systems into Modules," *Comm. ACM*, 15:12 (December), pp. 1053-1058.

PRIETO-DIAZ, R. and J.M. NEIGHBORS <1986>. "Module Interconnection Languages," *J. Systems and Software*, 6:4 (November), pp. 307-334. Reprinted in Freeman <1987>.

PRIETO-DIAZ, R. and P. FREEMAN <1987>. "Classifying Software for Reusability," *IEEE Software* (January), pp. 6-16. Reprinted in Freeman <1987>.

TSAI, Y. <1987>. "An Operational Approach to Model Integration Using a Structured Modeling Framework," Research Paper, Anderson Graduate School of Management, UCLA, 60 pages, December. Revised 2/88.

VICUÑA, F. <1988>. "A Modeling Environment for Operations Research," Ph.D. Research Proposal, UCLA, July.

YOURDON, E. and L.L. CONSTANTINE <1979>. *Structured Design*, Prentice-Hall, Englewood Cliffs, NJ.

ZEIGLER, B. P. <1984>. *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, London.

Appendix 1

FIRST EXAMPLE: FCAST + TRANS2

TRANS2 (see Appendix 4) is the classical Hitchcock-Koopmans transportation model and FCAST (see Appendix 5) is a simple exponential smoothing model. Our aim is to reuse these by integration so that the demands in TRANS2 are produced by FCAST.

Notice that FCAST is designed to forecast one item, not several. So it is necessary to make a multi-item forecasting model out of FCAST. This means replacing PROD by CUST in FCAST, with quite a few corresponding induced changes. This is a structural change because it introduces a new index. Assume that the same ALPHA is used for each CUSTOMER. Also, we must redefine F so that it produces a forecast only for the last time period. This is also a structural change. Rename F to DEMF and revise its defined key phrase accordingly. Rename the root module to &FORECAST.

The result of Step 1 applied to FCAST is as follows. Step 1 does nothing to TRANS2 except to rename the root module to &TRANSP.

&FORECAST FORECASTING SECTOR

TIME t /pe/ 3 <= Size {TIME} There is a list of TIME periods (at least 3 of them).

CUSTj /pe/ There is a list of CUSTOMERS.

ALPHA (CUST) /a/ : 0 <= Real <= 1 There is a SMOOTHING CONSTANT to be used for all CUSTOMERS.

D (CUSTj, TIMEt) /a/ {CUST} x {TIME} For each CUSTOMER and TIME period there is a DEMAND level.

E (ALPHA, Dj<1:t>) /f/ {CUST} x {TIME} ; @IF(Ord(t)>1, @SUMt'<2:t> (ALPHA * ((1-ALPHA)^(Ord(t)-Ord(t'))) * Djt'), 0) + ((1-ALPHA)^(Ord(t)-1)) * Dj<1> For each CUSTOMER and TIME period there is an EXPONENTIAL AVERAGE based on smoothed DEMAND history up to and including the TIME period at hand.

Appendix 1: Fcast + TRans2

S (ALPHA, Ej<t-2:t>) /f/ {CUST} x (Filter(t>2) {TIME}) ; ALPHA * (Ejt - Ej<t-1>) + (1-ALPHA) * (Ej<t-1> - Ej<t-2>) For each CUSTOMER and TIME period after the second there is a SMOOTHED TREND based on the EXPONENTIAL AVERAGES for the TIME period at hand and the two before that.

DEMF (ALPHA, Ej<-1>, Sj<-1>) /f/ {CUST} ; Ej<-1> + Sj<-1>/ALPHA For each CUSTOMER there is a one-ahead DEMAND FORECAST.

Step 2a. The key phrase DEMAND is not unique; change the one in &FORECAST to HISTORICAL DEMAND.

Step 2b. The merging genus pairs are (CUST in &FORECAST, CUST in &TRANSP) and (DEMF in &FORECAST, DEM in &TRANSP). The first pair of genus paragraphs is already identical, since we imported CUST into &FORECAST. The second pair cannot be made identical because DEM would have to become DEMF, which is not possible since DEMF calls some genera that are not in &TRANSP (remember, the schemas have not been joined yet).

Step 2c. Put DEMF into its own output module and annotate &CDATA that it is an input module.

Step 3a. Concatenate &FORECAST and &TRANSP in that order.

Step 3b. Replace the DEM paragraph in &TRANSP by the DEMF paragraph. Be sure to change all calls to it. Drop CUST and DEMF in &CDATA.

Step 3c. Null.

The result is as follows (the new root module is not shown).

&FORECAST FORECASTING SECTOR

TIME t /pe/ 3 <= Size {TIME} There is a list of TIME periods (at least 3 of them).

CUSTj /pe/ There is a list of CUSTOMERS.

ALPHA (CUST) /a/ : 0 <= Real <= 1 There is a SMOOTHING CONSTANT to be used for all CUSTOMERS.

D (CUSTj, TIMEt) /a/ {CUST} x {TIME} For each CUSTOMER and TIME period there is a HISTORICAL DEMAND level.

```
    E (ALPHA, Dj<1:t>) /f/ {CUST} x {TIME} ; @IF(Ord(t)>1,
    @SUMt'<2:t> (ALPHA * ((1-ALPHA)^(Ord(t)-Ord(t'))) * Djt'), 0)
    + ((1-ALPHA)^(Ord(t)-1)) * Dj<1>  For each CUSTOMER and TIME
    period there is an EXPONENTIAL AVERAGE based on smoothed
    HISTORICAL DEMAND history up to and including the TIME period
    at hand.

    S (ALPHA, Ej<t-2:t>) /f/ {CUST} x (Filter(t>2) {TIME}) ; ALPHA
    * (Ejt - Ej<t-1>) + (1-ALPHA) * (Ej<t-1> - Ej<t-2>)  For each
    CUSTOMER and TIME period after the second there is a SMOOTHED
    TREND based on the EXPONENTIAL AVERAGES for the TIME period at
    hand and the two before that.

    &F   FORECAST  (output module)

       DEMF (ALPHA, Ej<-1>, Sj<-1>) /f/ {CUST} ; Ej<-1> + Sj<-1> /
       ALPHA  For each CUSTOMER there is a one-ahead DEMAND FORECAST.

&TRANSP  TRANSPORTATION SECTOR

    &SDATA  SOURCE DATA

       PLANTi /pe/  There is a list of PLANTS.

       SUP (PLANTi) /a/ {PLANT} : Real+  Every PLANT has a SUPPLY
       CAPACITY measured in tons.

    &CDATA  CUSTOMER DATA (input module)

    &TDATA  TRANSPORTATION DATA

       LINK (PLANTi,CUSTj) /ce/ Select {PLANT} x {CUST} where i
       covers {PLANT}, j covers {CUST}  There are some
       transportation LINKS from PLANTS to CUSTOMERS. There must
       be at least one LINK incident to each PLANT, and at least
       one LINK incident to each CUSTOMER.

       FLOW (LINKij) /va/ {LINK} : Real+  There can be a
       nonnegative transportation FLOW (in tons) over each LINK.

       COST (LINKij) /a/ {LINK}  Every LINK has a TRANSPORTATION
       COST RATE for use in $/ton.

    $ (COST,FLOW) /f/ 1 ; @SUMi SUMj (COSTij * FLOWij)  There is
    a TOTAL COST associated with all FLOWS.
```

T:SUP (FLOWi.,SUPi) /t/ {PLANT} ; @SUMj (FLOWij) <= SUPi
Is the total FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the <u>SUPPLY TEST</u>.

T:DEM (FLOW.j,DEMFj) /t/ {CUST} ; @SUMi (FLOWij) = DEMFj
Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND FORECAST? This is called the <u>DEMAND TEST</u>.

Step 4. It is only necessary to delete the degenerate module &CDATA.

Step 5. This step can be null. Alternatively, one could restore the &CDATA module, move CUST inside it, and move the entire remains of the &FORECAST module inside of it right after CUST. The modular outline of the result would be as follows. (The complete schema can be found in Geoffrion <1988c> under the name T_FCAST.)

```
&TRANSP  TRANSPORTATION SECTOR
   &SDATA  SOURCE DATA
      PLANT
      SUP
   &CDATA  CUSTOMER DATA (input module)
      CUST
      &FORECAST  FORECASTING SECTOR
         TIME
         ALPHA
         D
         E
         S
         &F  FORECAST
            DEMF
   &TDATA  TRANSPORTATION DATA
      LINK
      FLOW
      COST
   $
   T:SUP
   T:DEM
```

An interesting observation is that this schema can be viewed as a <u>refinement</u> of the original TRANS2 schema rather than an integration of it with FCAST; of course, the reconciliation of these two viewpoints is that the refinement has been accomplished via integration.

Appendix 2

SECOND EXAMPLE: TRANS2 + MEOQ1

This example is the one treated in Section 3.2 of Geoffrion <1987a>. The discussion there was at a fairly high level of abstraction; here we consider details according to the 5-step approach suggested in Section 2. The motivation for this example will not be repeated. See Appendix 4 for the TRANS2 schema and Appendix 6 for the MEOQ1 schema. See also Tsai <1987>, who treats the same example in detail.

Step 1a. Null.

Step 1b. Rename both root modules. Rename SETUP$ to REC$ and change the word "setup" to the word "receipt" in several paragraphs. It would be desirable to do more customization, but not much more can be done without referencing one schema in the other. Change "units" to "tons" in the second schema for consistency with the first. No structural changes are necessary.

Step 2a. Null. There is a duplicate index; this conflict could be eliminated, but the work to do this would be undone at Step 3b.

Step 2b. The merging genus pairs are (LINK,ITEM) and (FLOW,D). Nothing can be done (without calling the first schema from the second) to make them more similar.

Step 2c. Put LINK and FLOW in &OUTPUT and put ITEM and D in &INPUT.

The two schemas after Step 2 are as follows.

&TRANS <u>TRANSPORTATION SECTOR</u>

&SDATA <u>SOURCE DATA</u>

PLANTi /pe/ There is a list of <u>PLANTS</u>.

SUP (PLANTi) /a/ {PLANT} : **Real+** Every PLANT has a <u>SUPPLY CAPACITY</u> measured in tons.

&CDATA <u>CUSTOMER DATA</u>

CUSTj /pe/ There is a list of <u>CUSTOMERS</u>.

DEM (CUSTj) /a/ {CUST} : Real+ Every CUSTOMER has a nonnegative DEMAND measured in tons.

&TDATA TRANSPORTATION DATA

&OUTPUT

LINK (PLANTi,CUSTj) /ce/ Select {PLANT} x {CUST} where i covers {PLANT}, j covers {CUST} There are some transportation LINKS from PLANTS to CUSTOMERS. There must be at least one LINK incident to each PLANT, and at least one LINK incident to each CUSTOMER.

FLOW (LINKij) /va/ {LINK} : Real+ There can be a nonnegative transportation FLOW (in tons) over each LINK.

COST (LINKij) /a/ {LINK} Every LINK has a TRANSPORTATION COST RATE for use in $/ton.

$ (COST,FLOW) /f/ 1 ; @SUMi SUMj (COSTij * FLOWij) There is a TOTAL COST associated with all FLOWS.

T:SUP (FLOWi.,SUPi) /t/ {PLANT} ; @SUMj (FLOWij) <= SUPi Is the total FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST.

T:DEM (FLOW.j,DEMj) /t/ {CUST} ; @SUMi (FLOWij) = DEMj Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the DEMAND TEST.

&EOQ EOQ SECTOR

&ITEMDATA Certain ITEM DATA are provided.

&INPUT

ITEMi /pe/ There is a list of ITEMS.

D (ITEMi) /a/ {ITEM} : Real+ Every ITEM has a DEMAND RATE (tons per year).

H (ITEMi) /a/ {ITEM} : Real+ Every ITEM has a HOLDING COST RATE (dollars per ton per year).

F (ITEMi) /a/ {ITEM} : Real+ Every ITEM has a <u>FIXED RECEIPT COST</u> (dollars per receipt).

Q (ITEMi) /va/ {ITEM} : Real+ The <u>ORDER QUANTITY</u> (tons per order) for each ITEM is to be chosen.

&OPCON <u>OPERATING CONSEQUENCES</u> of ORDER QUANTITY choices.

FREQ (Di,Qi) /f/ {ITEM} ; Di / Qi Every ITEM has a <u>RECEIPT FREQUENCY</u> (average number of receipts per year) equal to DEMAND RATE divided by ORDER QUANTITY.

REC$ (FREQi,Fi) /f/ {ITEM} ; FREQi * Fi Every ITEM has an <u>ANNUAL RECEIVING COST</u> (dollars per year) equal to the RECEIPT FREQUENCY times the FIXED RECEIPT COST.

CARRY$ (Hi,Qi) /f/ {ITEM} ; Hi * Qi / 2 Every ITEM has an <u>ANNUAL CARRYING COST</u> (dollars per year) equal to its HOLDING COST RATE times one-half of its ORDER QUANTITY (which estimates average inventory level).

ITEM$ (REC$i,CARRY$i) /f/ {ITEM} ; REC$i + CARRY$i Every ITEM has an <u>ANNUAL ITEM COST</u> (dollars per year) equal to its ANNUAL RECEIPT COST plus its ANNUAL CARRYING COST.

TOT$ (ITEM$) /f/ 1 ; @SUMi (ITEM$i) The <u>TOTAL ANNUAL COST</u> (dollars per year) is the sum of all ANNUAL ITEM COSTS.

Step 3a. Concatenate, with &TRANS coming first.

Step 3b. Change ITEM to LINK and D to FLOW, with all the ramifications that this entails. Drop the second replicates.

Step 3c. Null.

The result is as follows (the new root module is not shown).

&TRANS <u>TRANSPORTATION SECTOR</u>

&SDATA <u>SOURCE DATA</u>

PLANTi /pe/ There is a list of <u>PLANTS</u>.

SUP (PLANTi) /a/ {PLANT} : Real+ Every PLANT has a <u>SUPPLY CAPACITY</u> measured in tons.

&CDATA <u>CUSTOMER DATA</u>

CUSTj /pe/ There is a list of <u>CUSTOMERS</u>.

DEM (CUSTj) /a/ {CUST} : Real+ Every CUSTOMER has a nonnegative <u>DEMAND</u> measured in tons.

&TDATA <u>TRANSPORTATION DATA</u>

&OUTPUT

LINK (PLANTi,CUSTj) /ce/ Select {PLANT} x {CUST} where i covers {PLANT}, j covers {CUST} There are some transportation <u>LINKS</u> from PLANTS to CUSTOMERS. There must be at least one LINK incident to each PLANT, and at least one LINK incident to each CUSTOMER.

FLOW (LINKij) /va/ {LINK} : Real+ There can be a nonnegative transportation <u>FLOW</u> (in tons) over each LINK.

COST (LINKij) /a/ {LINK} Every LINK has a <u>TRANSPORTATION COST RATE</u> for use in $/ton.

$ (COST,FLOW) /f/ 1 ; @SUMi SUMj (COSTij * FLOWij) There is a <u>TOTAL COST</u> associated with all FLOWS.

T:SUP (FLOWi.,SUPi) /t/ {PLANT} ; @SUMj (FLOWij) <= SUPi Is the total FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the <u>SUPPLY TEST</u>.

T:DEM (FLOW.j,DEMj) /t/ {CUST} ; @SUMi (FLOWij) = DEMj Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the <u>DEMAND TEST</u>.

&EOQ <u>EOQ SECTOR</u>

&ITEMDATA Certain <u>ITEM DATA</u> are provided.

&INPUT

H (LINKij) /a/ {LINK} : Real+ Every LINK has a <u>HOLDING COST RATE</u> (dollars per ton per year).

F (LINKij) /a/ {LINK} : Real+ Every LINK has a <u>FIXED RECEIPT COST</u> (dollars per receipt).

Q (LINKij) /va/ {LINK} : Real+ The <u>ORDER QUANTITY</u> (tons per order) for each LINK is to be chosen.

&OPCON <u>OPERATING CONSEQUENCES</u> of ORDER QUANTITY choices.

FREQ (FLOWij,Qij) /f/ {LINK} ; FLOWij / Qij Every LINK has a <u>RECEIPT FREQUENCY</u> (average number of receipts per year) equal to FLOW divided by ORDER QUANTITY.

REC$ (FREQij,Fij) /f/ {LINK} ; FREQij * Fij Every LINK has an <u>ANNUAL RECEIVING COST</u> (dollars per year) equal to the RECEIPT FREQUENCY times the FIXED RECEIPT COST.

CARRY$ (Hij,Qij) /f/ {LINK} ; Hij * Qij / 2 Every LINK has an <u>ANNUAL CARRYING COST</u> (dollars per year) equal to its HOLDING COST RATE times one-half of its ORDER QUANTITY (which estimates average inventory level).

ITEM$ (REC$ij,CARRY$ij) /f/ {LINK} ; REC$ij + CARRY$ij Every LINK has an <u>ANNUAL LINK COST</u> (dollars per year) equal to its ANNUAL RECEIPT COST plus its ANNUAL CARRYING COST.

TOT$ (ITEM$) /f/ 1 ; @SUMi SUMj (ITEM$ij) The <u>TOTAL ANNUAL COST</u> (dollars per year) is the sum of all ANNUAL LINK COSTS.

Step 4. Drop the degenerate module &INPUT.

Step 5. Change the name of $ to TRANS$, ITEM$ to LINK$, and TOT$ to INV$ for cosmetic reasons; change the associated key phrases also. Create a new genus TOTCOST to sum up TRANS$ and INV$; put it at the very end as a sibling of the two first level modules. The complete schema can be found in Geoffrion <1988c> under the name T_MEOQ.

Appendix 3

THIRD EXAMPLE: TRANS2 + TRANS2

The aim is to take two classical transportation models and connect them so that the output of the first becomes the input to the second. The result will be a two-echelon transshipment problem. See Appendix 4 for the TRANS2 schema. Incidentally, this is the same example as the one worked in Bradley and Clemence <1988>.

Step 1a. Copy TRANS2 to the workspace. Rename its root module to "&FIRST_ECHELON". Specialize its structure to its intended use as the first echelon: drop DEM and T:DEM (outputs will go to the 2nd echelon). Copy TRANS2 to the workspace. Rename its root module to "&SECOND_ECHELON". Specialize its structure to its intended use as the second echelon: drop SUP and T:SUP (inputs will come from the 1st echelon).

Step 1b. In &FIRST_ECHELON: rename CUST to DC, LINK to IBLINK, &TDATA to &IBDATA, FLOW to IBFLOW, COST to IBCOST; and change key phrases accordingly. In &SECOND_ECHELON: rename PLANT to DC, LINK to OBLINK, &TDATA to &OBDATA, FLOW to OBFLOW, COST to OBCOST; and change key phrases accordingly. Change the key phrase for $ in both schemas. Change the name of DC's module to &DCDATA in both schemas; change the corresponding key phrases to match.

The two schemas at this point are as follows.

&FIRST_ECHELON <u>FIRST ECHELON</u>

&SDATA <u>SOURCE DATA</u>

PLANTi /pe/ There is a list of <u>PLANTS</u>.

SUP (PLANTi) /a/ {PLANT} : Real+ Every PLANT has a <u>SUPPLY CAPACITY</u> measured in tons.

&DCDATA <u>DC DATA</u>

DCj /pe/ There is a list of <u>DCS</u>.

&IBDATA <u>INBOUND TRANSPORTATION DATA</u>

IBLINK (PLANTi,DCj) /ce/ Select {PLANT} x {DC} where i covers {PLANT}, j covers {DC} There are some transportation <u>INBOUND LINKS</u> from PLANTS to DCS. There must be at least one INBOUND LINK incident to each PLANT, and at least one INBOUND LINK incident to each DC.

IBFLOW (IBLINKij) /va/ {IBLINK} : Real+ There can be a nonnegative transportation <u>INBOUND FLOW</u> (in tons) over each INBOUND LINK.

IBCOST (IBLINKij) /a/ {IBLINK} Every INBOUND LINK has an <u>INBOUND TRANSPORTATION COST RATE</u> for use in $/ton.

$ (IBCOST,IBFLOW) /f/ 1 ; @SUMi SUMj (IBCOSTij * IBFLOWij) There is a <u>TOTAL INBOUND COST</u> associated with all INBOUND FLOWS.

T:SUP (IBFLOWi.,SUPi) /t/ {PLANT} ; @SUMj (IBFLOWij) <= SUPi Is the total INBOUND FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the <u>SUPPLY TEST</u>.

&SECOND_ECHELON <u>SECOND ECHELON</u>

&DCDATA <u>DC DATA</u>

DCi /pe/ There is a list of <u>DCS</u>.

&CDATA <u>CUSTOMER DATA</u>

CUSTj /pe/ There is a list of <u>CUSTOMERS</u>.

DEM (CUSTj) /a/ {CUST} : Real+ Every CUSTOMER has a nonnegative <u>DEMAND</u> measured in tons.

&OBDATA <u>OUTBOUND TRANSPORTATION DATA</u>

OBLINK (DCi,CUSTj) /ce/ Select {DC} x {CUST} where i covers {DC}, j covers {CUST} There are some transportation <u>OUTBOUND LINKS</u> from DCS to CUSTOMERS. There must be at least one OUTBOUND LINK incident to each DC, and at least one OUTBOUND LINK incident to each CUSTOMER.

OBFLOW (OBLINKij) /va/ {OBLINK} : Real+ There can be a nonnegative transportation <u>OUTBOUND FLOW</u> (in tons) over each OUTBOUND LINK.

OBCOST (OBLINKij) /a/ {OBLINK} Every OUTBOUND LINK has an <u>OUTBOUND TRANSPORTATION COST RATE</u> for use in $/ton.

$ (OBCOST,OBFLOW) /f/ 1 ; @SUMi SUMj (OBCOSTij * OBFLOWij) There is a <u>TOTAL OUTBOUND COST</u> associated with all OUTBOUND FLOWS.

T:DEM (OBFLOW.j,DEMj) /t/ {CUST} ; @SUMi (OBFLOWij) = DEMj Is the total OUTBOUND FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the <u>DEMAND TEST</u>.

Step 2a. There is a pair of genera with the duplicated name $; change to IB$ and OB$ respectively. There are two pairs of indices with duplicate names: i and j. Change j in &SECOND_ ECHELON to k, and change i in &SECOND_ECHELON to j since that will serve the purposes of Step 2b. There are two identical module paragraphs named &DCDATA. It does no harm to leave them as is.

Step 2b. The two schemas will be merged on the genus pair (DC,DC). Both genus paragraphs are already identical.

Step 2c. Skip this option.

Step 3a. Concatenate in the order &FIRST_ECHELON, &SECOND_ECHELON.

Step 3b. The merging genus pair is already identical. Drop the second DC genus.

Step 3c. Introduce a material balance genus T:DC that checks whether total inflow equals total outflow at the DCs.

The integrated schema is as follows.

&FIRST_ECHELON <u>FIRST ECHELON</u>

&SDATA <u>SOURCE DATA</u>

PLANTi /pe/ There is a list of <u>PLANTS</u>.

SUP (PLANTi) /a/ {PLANT} : Real+ Every PLANT has a <u>SUPPLY CAPACITY</u> measured in tons.

&DCDATA DC DATA

DCj /pe/ There is a list of DCS.

&IBDATA INBOUND TRANSPORTATION DATA

IBLINK (PLANTi,DCj) /ce/ Select {PLANT} x {DC} where i covers {PLANT}, j covers {DC} There are some transportation INBOUND LINKS from PLANTS to DCS. There must be at least one INBOUND LINK incident to each PLANT, and at least one INBCUND LINK incident to each DC.

IBFLOW (IBLINKij) /va/ {IBLINK} : Real+ There can be a nonnegative transportation INBOUND FLOW (in tons) over each INBOUND LINK.

IBCOST (IBLINKij) /a/ {IBLINK} Every INBOUND LINK has an INBOUND TRANSPORTATION COST RATE for use in $/ton.

IB$ (IBCOST,IBFLOW) /f/ 1 ; @SUMi SUMj (IBCOSTij * IBFLOWij) There is a TOTAL INBOUND COST associated with all INBOUND FLOWS.

T:SUP (IBFLOWi.,SUPi) /t/ {PLANT} ; @SUMj (IBFLOWij) <= SUPi Is the total INBOUND FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST.

&SECOND_ECHELON SECOND ECHELON

&DCDATA DC DATA

&CDATA CUSTOMER DATA

CUSTk /pe/ There is a list of CUSTOMERS.

DEM (CUSTk) /a/ {CUST} : Real+ Every CUSTOMER has a nonnegative DEMAND measured in tons.

&OBDATA OUTBOUND TRANSPORTATION DATA

OBLINK (DCj,CUSTk) /ce/ Select {DC} x {CUST} where j covers {DC}, k covers {CUST} There are some transportation OUTBOUND LINKS from DCS to CUSTOMERS. There must be at least one OUTBOUND LINK incident to each DC, and at least one OUTBOUND LINK incident to each CUSTOMER.

OBFLOW (OBLINKjk) /va/ {OBLINK} : Real+ There can be a nonnegative transportation <u>OUTBOUND FLOW</u> (in tons) over each OUTBOUND LINK.

OBCOST (OBLINKjk) /a/ {OBLINK} Every OUTBOUND LINK has an <u>OUTBOUND TRANSPORTATION COST RATE</u> for use in $/ton.

OB$ (OBCOST,OBFLOW) /f/ 1 ; @SUMj SUMk (OBCOSTjk * OBFLOWjk) There is a <u>TOTAL OUTBOUND COST</u> associated with all OUTBOUND FLOWS.

T:DEM (OBFLOW.k,DEMk) /t/ {CUST} ; @SUMj (OBFLOWjk) = DEMk Is the total OUTBOUND FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the <u>DEMAND TEST</u>.

T:DC (IBFLOW.j,OBFLOWj.) /t/ {DC} ; @SUMi (IBFLOWij) = @SUMk (OBFLOWjk) Is the total INBOUND FLOW arriving at each DC exactly equal to the total OUTBOUND FLOW leaving it? This is called the <u>DC MATERIAL BALANCE TEST</u>.

Step 4. Delete the vestigial module &DCDATA in &SECOND_ECHELON.

Step 5. Add a new genus TOT$ to add up the two cost subtotals. Make it a sibling of &FIRST_ECHELON and &SECOND_ECHELON. This paragraph is:

TOT$ (IB$,OB$) /f/ 1 ; IB$ + OB$ The <u>TOTAL COST</u> is the sum of the TOTAL INBOUND COST and TOTAL OUTBOUND COST.

The complete schema can be found in Geoffrion <1988c> under the name TRANS_2E.

Appendix 4: *CLASSICAL TRANSPORTATION MODEL*

&SDATA SOURCE DATA

PLANTi /pe/ There is a list of PLANTS.

SUP (PLANTi) /a/ {PLANT} : Real+ Every PLANT has a SUPPLY CAPACITY measured in tons.

&CDATA CUSTOMER DATA

CUSTj /pe/ There is a list of CUSTOMERS.

DEM (CUSTj) /a/ {CUST} : Real+ Every CUSTOMER has a nonnegative DEMAND measured in tons.

&TDATA TRANSPORTATION DATA

LINK (PLANTi,CUSTj) /ce/ Select {PLANT} x {CUST} where i covers {PLANT}, j covers {CUST} There are some transportation LINKS from PLANTS to CUSTOMERS. There must be at least one LINK incident to each PLANT, and at least one LINK incident to each CUSTOMER.

FLOW (LINKij) /va/ {LINK} : Real+ There can be a nonnegative transportation FLOW (in tons) over each LINK.

COST (LINKij) /a/ {LINK} Every LINK has a TRANSPORTATION COST RATE for use in $/ton.

$ (COST,FLOW) /f/ 1 ; @SUMi SUMj (COSTij * FLOWij) There is a TOTAL COST associated with all FLOWS.

T:SUP (FLOWi.,SUPi) /t/ {PLANT} ; @SUMj (FLOWij) <= SUPi Is the total FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST.

T:DEM (FLOW.j,DEMj) /t/ {CUST} ; @SUMi (FLOWij) = DEMj Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the DEMAND TEST.

Appendix 5: *EXPONENTIAL SMOOTHING MODEL*

TIME t /pe/ 3 <= Size {TIME} There is a list of <u>TIME</u> periods (at least 3 of them).

PROD /pe/ There is a <u>PRODUCT</u> whose demand process is to be forecasted.

ALPHA (PROD) /a/ : 0 <= Real <= 1 There is a <u>SMOOTHING CONSTANT</u> to be used for the PRODUCT.

D (PROD, TIMEt) /a/ {TIME} For each TIME period there is a <u>DEMAND</u> level for the PRODUCT.

E (ALPHA, D<1:t>) /f/ {TIME} ; @IF(Ord(t)>1, @SUMt'<2:t> (ALPHA * ((1-ALPHA)^(Ord(t)-Ord(t'))) * Dt'), 0) + ((1-ALPHA)^(Ord(t)-1)) * D<1> For each TIME period there is an <u>EXPONENTIAL AVERAGE</u> based on smoothed DEMAND history up to and including the TIME period at hand. < Note: The generic rule for t>1 is just the fully expanded version of ALPHA*Dt + (1-ALPHA)*Et-1. We should consider expanding the generic rule syntax to allow this sort of thing, since it does not really violate acyclicity. Also, the @IF function is necessary because Et has a special initialization formula for t=1; recall that @SUMt'<2:t> includes the (unwanted) t'=2 term, and only this term, when t=1. >

S (ALPHA, E<t-2:t>) /f/ Filter(t>2) {TIME} ; ALPHA * (Et - E<t-1>) + (1-ALPHA) * (E<t-1> - E<t-2>) For each TIME period after the second there is a <u>SMOOTHED TREND</u> based on the EXPONENTIAL AVERAGES for the TIME period at hand and the two before that.

F (ALPHA, Et, St) /f/ Filter(t>2) {TIME} ; Et + St/ALPHA For each TIME period after the second there is a one-ahead <u>FORECAST</u> of DEMAND. < Note: It is tempting to call TIME<t+1> in the calling sequence, but this would create a problem for the last time period. It can be overcome by excluding the last time period from the ISS, but then D, E, and S would have to be unspecified for the last time period if this period really represents the future. So the ISS of those genera would have to exclude the last period. All this works out fine in structured modeling terms, but for now we simply leave the formulation as presently given. >

Appendix 6: *MULTI-ITEM EOQ MODEL*

ITEMi /pe/ There is a list of ITEMS.

&ITEMDATA Certain ITEM DATA are provided.

D (ITEMi) /a/ {ITEM} : Real+ Every ITEM has a DEMAND RATE (units per year).

H (ITEMi) /a/ {ITEM} : Real+ Every ITEM has a HOLDING COST RATE (dollars per unit per year).

F (ITEMi) /a/ {ITEM} : Real+ Every ITEM has a FIXED SETUP COST (dollars per setup).

Q (ITEMi) /va/ {ITEM} : Real + The ORDER QUANTITY (units per order) for each ITEM is to be chosen.

&OPCON OPERATING CONSEQUENCES of ORDER QUANTITY choices.

FREQ (Di,Qi) /f/ {ITEM} ; Di / Qi Every ITEM has a SETUP FREQUENCY (average number of setups per year) equal to DEMAND RATE divided by ORDER QUANTITY.

SETUP$ (FREQi,Fi) /f/ {ITEM} ; FREQi * Fi Every ITEM has an ANNUAL SETUP COST (dollars per year) equal to the SETUP FREQUENCY times the SETUP COST.

CARRY$ (Hi,Qi) /f/ {ITEM} ; Hi * Qi / 2 Every ITEM has an ANNUAL CARRYING COST (dollars per year) equal to its HOLDING COST RATE times one-half of its ORDER QUANTITY (which estimates average inventory level).

ITEM$ (SETUP$i,CARRY$i) /f/ {ITEM} ; SETUP$i + CARRY$i Every ITEM has an ANNUAL ITEM COST (dollars per year) equal to its ANNUAL SETUP COST plus its ANNUAL CARRYING COST.

TOT$ (ITEM$) /f/ 1 ; @SUMi (ITEM$i) The TOTAL ANNUAL COST (dollars per year) is the sum of all ANNUAL ITEM COSTS.